Computer Graphics
Prof. Sukhendu Das
Dept. of Computer Science and Engineering
Indian Institute of Technology, Madras
Lecture - 39
Graphics Programming using OpenGL

Hello and welcome everybody once again. Today we are going to discuss one of the most important and practical aspects of computer graphics. This is how you develop your own application programs or provided an API or an Application Programmers Interface how to draw pictures and simulate pictures on the screen. And as mentioned in the introduction of the lecture we are going to follow the industrial standard which is OpenGL or Open Graphics Library. So today the discussion will be on the Graphics Programming using OpenGL, that is where we start today.

(Refer Slide Time: 00:11:58)

The first discussion is; why OpenGL?
Well, there are a verity of reasons or various reasons why OpenGL has become a standard. And there has been a gradual transition from the old standards we talked of graphical standards in the introduction part of this course. And we started with core graphics moved on to SRGP PHIGS SPHIGS and finally the world community almost has accepted OpenGL as the current standard.

Of course things could change in the future but we will follow the standard OpenGL as of today which is available on various platforms. So coming back to the question why OpenGL, the first reason is device independence which you can see here. That is you should be able to run OpenGL independent of the device you are using.

That means today if you run your OpenGL program on a certain manufacturers keyboard, mouse and monitor and tomorrow if you move on to another system which almost has a entirely different platform, the mouse has changed to a different manufacturer, different type, different make, the keyboard, the mouse or even the processor or the operating system in certain cases. Then you should be able to run your program with almost no change.

Of course, when you change from one operating system to another it could be a case where you have to make some operating system related changes or compiler related changes may be but in terms of graphical calls, graphical functions which we have seen through all the theories which we have analyzed so far starting from transformations to clipping to rendering, shading and all those operations which we have studied so far, projection geometry let us say the command syntax and the command name in fact does not vary at all. That is the advantage which we have by sticking to a particular standard because if you if you do not follow a particular standard there are lots of difficulties which one could face.

The number one point is, nobody will be able to understand your program because you may make a program but it may be necessary for somebody else at a later point of time to alter the program, modify the program, enhance the programs capability, capacity and provided add on values. So if you do not follow any standard and write your own graphic routines and that is followed by a group of people in a computer graphics team and another is you will not be able to sure it with somebody else. So software maintenance, reuse and all those factors have a big role to play while we need corresponding graphic standards and device independence is very much welcome in this particular scenario.

Now the question comes is how do you use OpenGL? I did mention this in the introduction lecture that almost any operating system you name it which exist in the world starting from all varieties of windows environment to Linux to Irix to Sun Solaris or even Mac OS you should be able to download OpenGL drivers, DLLs, header files on your system which goes along not only with your operating system but also goes along with your driver cards.

Driver cards which are sitting on your system and once that is configured properly or what I will say as OpenGL ready. If the system is OpenGL ready you should be able to borrow or pick up any OpenGL program, open source program available or borrow it from your corresponding client or another group member and then straight away use it almost with no change or very little change.

You might have to make a very little change if you are moving from one system to another, so that is the device independence part of OpenGL which is discussed here and the first point in the slide which you have seen here. The number two is almost the same which is the platform independence we were talking about.

We are talking of various types of platforms starting from SGI Irix systems and Linux systems or Windows. Now the platform independence is different from device independence. Let us see the difference, why? What is the difference?

Well in the same platform it is possible that you could change your device. You could change your device, tomorrow the keyboard does not work, tomorrow the mouse does not work or you are not satisfied with the monitor or the monitors resolution. So what do you do? You can pick up another monitor in place of that in any of these cases. So that could happen with any system. And still you are you want your OpenGL programs to work on that. That is the device independence part or the characteristics or a feature which you want in this system.

When you talk of platform independence we are talking about the entire program being shunted out from one system to another and that is the portability when you talk about it. So you should able to take your programs from Linux to windows vise versa or take it to any other SGI Irix based systems and it should work. So that is platform independence. Device independence talks about interoperability from one device to another in terms of device components of a system whereas platform independence talks about interoperability over various platforms.

You want the programs to seamlessly hop from one machine to another and work straight away if possible. These are the two factors which dictates your need for OpenGL.

Well, we talk of these abstractions later on in OpenGL where we talk of a GL, GLU and a GLUT or gluts as it is called. GL stands for graphical library that is open so OpenGL stands for Open Graphics Library. GLU stands for Open Graphics Library Utility and what is GLUT then GLUT open I mean it is Graphics Library Utility Tool Kit. These are different levels of abstractions, one sits on top of the other certain high level features are available in GLUT which are not available in GLU. GLU has certain features not available in GL so we will see that later on.

I will bring in those three stages one by one in the later point of discussion where we are talking about base level constructs available in GL and a few more advanced features in GLU and GLUT is something like almost as a tool kit available to you which has a very high level interface with the windows GUI and things like that. This is what we will talk about, abstractions, later on when it comes to OpenGL.

Well, it is an open source that means you should be able to obtain vendor specific information not only in terms of the DLL device drivers, header files for your particular monitor and systems. Devices which are using in a platform but you can have open source course available. Open source course available which you can download and use and you can design your own DLLs if necessary for a certain device and it is free to use.

Although it is picked up as an industrial standard currently most universities also follow the current OpenGL syntax but it is an open source like Linux and Irix is freely available.

So that is what is OpenGL for you, it is an open source. And then we are talking about OpenGL as hardware independent software interface.

Well OpenGL is a software interface for any graphics hardware. That is the target of course there could be some few specific hardware components where OpenGL may not work. What I mean by do not work following that there may not be the corresponding dynamic link libraries or the DLLs or the header files available for that particular hardware component right now which are not ready to be interfaced and used. Otherwise it is a software interface for any graphical hardware that is the typical one.

An OpenGL's main purpose, where do you use OpenGL? Its main purpose is to render two dimensional or three dimensional objects. That is the main purpose why OpenGL is used; we will come to that point later on and find the different features in OpenGL. But as we go along this point we finish these points about why OpenGL and discuss it in detail about why do you need to use OpenGL.

So it is a hardware independent software interface to be used and it also supports the client server protocol. That means you can run it over a network when you talk of a client server protocol either using x Windows for Linux or you are talking of an environment over the Windows NT type of network client server interface where the OpenGL is expected to work. It supports the client servers protocol at least in terms of x windows protocol client servers model the OpenGL does not have a problem.

We assume here that when you have a client and a server of course in a traditional concept when the client sends a request to do some service and the server services that. When we are talking of an OpenGL server sitting which can handle all the graphics commands, all the graphics manipulations requested by the client and that client may not be the same machine, that processor may not be on the same machine but it could be a remote machine sitting on a network which is requesting the service and the OpenGL server serves that. So that sort of a support of client server protocol is also supported or taken care of in OpenGL.

So support of the client's server protocol is one important point. And the other point is we have the other application program as interface like the OpenInventor also by SGI which is an object oriented tool available. And of course you can have directex of Microsoft and java 3D of Sun Microsoft systems or alternatives of OpenGL which makes your life easy if required. So these are all toolkits and software interfaces available for OpenGL to work along with these. You do not have to worry too much about trying to interface, directex or java 3D along with the OpenGL they mesh quiet easily.

So the now the question comes with all these traditional work what are the features of OpenGL we will move towards next. But I hope for the time being the idea of why one uses OpenGL right now is very clear. Of course there have been standards earlier, we discussed about the old core graphics and we discussed about SRGP and PHIGS. All of them also had some of these features. There were some device dependent graphical libraries brought out by various vendors starting from HP to even sun in the earlier days.

But the problem we had was when the graphics program is coded by a programmer and you wanted to port it to another system which does not have those support of the libraries you could not easily make those programs run on any other system. That problem is almost completely removed by OpenGL and that is why people are quiet happy with these particular studies.

So the next question comes is what are the features about OpenGL. What OpenGL does and how it helps you. We will look at some of these features first about OpenGL and then see what you could do with this OpenGL and of course towards the end of the class if not today in the next class we will see what you cannot do also.

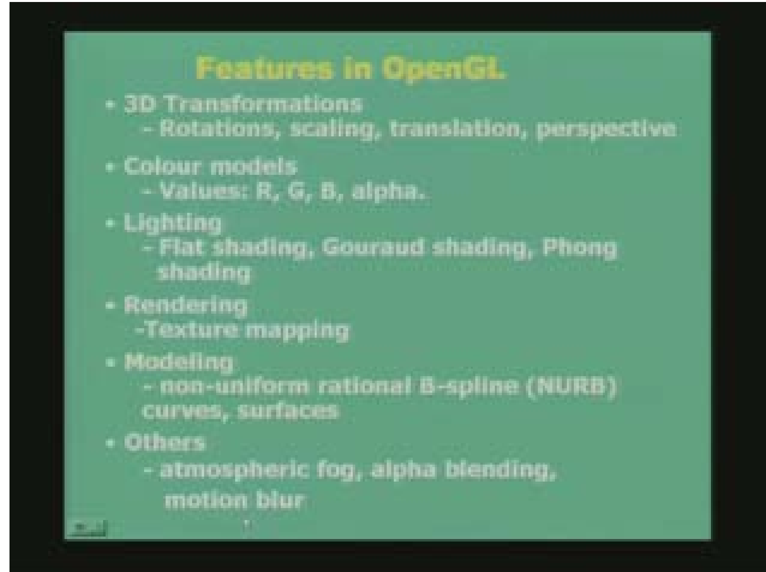
I must admit here and I must inform you also that although this is a lecture on software specific standards for graphics and it will give you only glimpses and highlights of what OpenGL is all about and what is possible for you to do. We will also see some examples in the next class with OpenGL code on a platform. Probably I will pick up Windows platform to show you were demonstration of OpenGL code.

But I would request you that if you when you have a system working you please download the OpenGL drivers they are available. Go to the vendor specific website or go to the standard OpenGL. org site. I will give you those links at the end of the class, next class if not today. And you should be able to download those device drivers, DLLs, header files and run your OpenGL.

Please run and practice, pick up a reference manual, pick up a programmers guide of OpenGL, take some sample codes, get experience and then try even complex programs so that you gain confidence. Remember, just listening to these lectures, looking to these slides, looking to the demo does not help you to learn.

It will probably trigger off an interest within you. it might just give you glimpses of how to start with OpenGL. You must pick up a programmer reference manual yourself make your system OpenGL ready. And I am sure you will be able to enjoy the features of OpenGL which are available world wide for any programmer. So let us look back into the features in OpenGL before we start to discuss about what is OpenGL.

(Refer Slide Time: 00:26:45)



It provides facilities such as 3D transformations which includes rotation, scaling, translation and perspective.

It provides color models; you have to specify value of the colors such as the red, green, blue, RGB and the alpha.

We will describe this alpha factor but before that we will go to certain features and also I would probably like to bring in here what is OpenGL in terms of trying to help you in drawing a picture. Well, the main purpose of OpenGL here is to provide you an interface such that which helps you to render, draw of course and then render both 2D and 3D objects as well as pixels in terms of bit map image into a frame buffer.

Ultimately OpenGL addresses a frame buffer. You do not have to worry about the frame buffer address and so on because all those are handled by OpenGL drivers themselves. But it provides you whatever functionalities we have studied so far in computer graphics if you recollect starting from transformations to clipping, drawing lines, polygons, circles, ellipses etc. We looked at good algorithms for VSD rendering and all those features which we have, all those features are available in OpenGL.

But of course, as a computer graphics scientist here and an engineer you must know how those algorithms have been drawn efficiently by OpenGL and that is why we went through all these algorithms as much as possible. But now you do not have to write the Bresenham's code in c or c plus plus.

A line draw equivalent comment of that is available in OpenGL in fact there are much more higher level commands to draw curves, polygon, poly lines or even shading. So the main purpose of OpenGL is to render 2D and 3D objects as well as draw pixels, as well as provide pixels on to the bit map screen into the frame buffer. And of course when you

talk of rendering 2D and 3D objects in most cases we will go through a flow chart and talk about the graphics pipeline in OpenGL.

But before that I want to give you a picture that you should be able to draw 2D and 3D objects in terms of vertices. So these vertices are the ones which are the key, which help you to draw lines starting from a single line to multiple lines to poly lines to polygons to curves, most efficient curves will introduce a curve, the term called nurbs, a closer to what we understood as B-splines earlier. So all these are treated as a set of vertices or what are even called as fragments. We will come across this term called fragments and these fragments process to a stage of an OpenGL pipeline.

You can visualize OpenGL pipeline similar to the 3D graphics rendering pipeline which we have studied earlier in our viewing pipeline. So you can visualize almost all that. We will study about the viewing pipeline exclusively and we will just remember here that OpenGL draws primitives. And when those primitives are drawn you should be able to operate on primitives.

What are those operations? Some of these features are given here, we talk of 3D transformations such as rotation, scaling, translation, perspective etc. You can put a color on to those primitives, so OpenGL draws primitives.

It draws points, lines or polygons, of course it has several other selectable modes. You can control a mode which is not given here in terms of color could be visualized as a mode in the sense that when you enter a certain mode, of course there could be several modes operating in OpenGL and some of them do not interact with each other in terms of a color may not interact with animations mode set by you or you could have two different parts having two different colors of a segment.

But in certain cases the mode does interact depending upon the situation. So one must have an idea what mode interacts with the other and what mode does not interact. You can almost visualize that. Those with computer science background must, I can speak to them here, and you can visualize OpenGL almost as a state machine where you move on to a certain state and the state remains still you change that state.

Now, let us look at a color. If you set a color default to your background, foreground could be red, blue, green and magenta whatever it is, we talked of these 8 fundamental colors with 3 bits or 256 possible colors with 8 bit and so on. You remember, the concept of look-up-table when we discussed about graphics display. So you can set a particular color. And when you set a particular color and keep drawing primitives such as points, lines, vertices, poly lines.

Remember, these are the primitives, very low level primitives, you can of course use an object oriented approach and encapsulate and draw high level primitives which will over ride on the set of low level primitives provided by OpenGL and all that is possible. But when you are into a mode with a certain color, all those primitives will be drawn with that particular color unless you change that mode.

Unless you change that mode and move to another color, let us say in this case I am talking of color as an example of a mode but it could be something else as well. When you set a particular color that mode is particular and is set for all primitives run after that till you change the color then you have basically gone on to a different mode.

So, we will look at these features, we will continue with the discussion 3D transformations, color then lighting which provide facilities for flat shading, Gouraud shading, Phong shading etc. And it also provides rendering in terms of texture mapping which is a unique feature in OpenGL where you can map on to a texture on to a particular surface. And of course modeling, here we are talking about primitives to be defined as a group of one or more vertices.

We talked of primitives as points, vertices, lines, you can have primitives as curves. A vertex could be defined as a point or the end point of a line or a corner or a polygon and things like that. A primitive could be a curvature. And when you look at this particular curve which we called as nurbs is a very new term which I am introducing here with respect to OpenGL. This concept of nurbs is available in high level toolkits such as maya. I gave this example of the software maya in the introduction of the lecture which is used for making movies.

Good animation high quality virtual reality movies are made of maya and maya provides you to create complex objects not only with points, lines, polygons but it also provides you nurbs.

What is nurbs? It is a non uniform rational B spline. I repeat, remember, you can note down this term non uniform rational B spline. We talked about cubic spline and Bezier curves, a combination of that whereas a spline uses the Bernstein basis called a B spline. So we have this nurbs has curves and surfaces and that could be also used to define your primitives. And the data which it could consist of vertices, colors, surface normal, texture coordinate, edge flags, which could be associated with either a vertex or a particular curve or a primitive in its associated data they are processed independently and they are processed in the graphics pipeline.

The graphics pipeline will process all the attributes necessary for the particular primitive. And I have used the word primitive earlier in the lecture series and I use it again here in OpenGL. the word primitive does exist and I repeat the primitive could be a single point, it could be a single line, it could be a group of lines, it could be a polygon, it could be a shaded polygon or a poly line, it could be a nurb, it could be a simple triangle, it could be a surface and there are various attributes associated with these primitives.

Of course the first fundamental descriptors you need for defining primitives at their x y z coordinates is one point or multiple points. Then depending upon the type of a primitive you could have its surface normal, you could have its color, you could have its texture, you could have other properties. We will see later on like the material properties of a surface which could be used for rendering and shading. So, all these are very important

properties which are associated with lightning and shading which we discussed also in the Phong model and the Gouraud model also. Those are important features which must be incorporated as attributes of your primitives.

And their exist an OpenGL graphics pipeline which will process this data, data means a stream of such request to process these set of vertices in a certain sequence, not randomly of course, is when a sequence of events starts in OpenGL.

When a sequence opens, events start it is an execution model which takes over and it starts interpreting your OpenGL code and it could move from one mode to another and then it starts processing the sequence of vertices the way in which your program is written. So we are talking of features of openGL which work on a set of primitives defined by your program and these are the features. We talked about 3D transformations. If you read through the slide, color models, lightning you can define more than one light source, you can have various types of lightning properties, and you can define a light based on a certain color. And the surface material also can have a certain color we already discussed about this where the light color could be different than the color of the surface material. And of course they could interact to give you different phenomena.

The lightning involves various other properties like it can also talk about the effect of attenuations, the atmospheric attenuation of what is called as the term used in OpenGL called fog, the effect of fog.

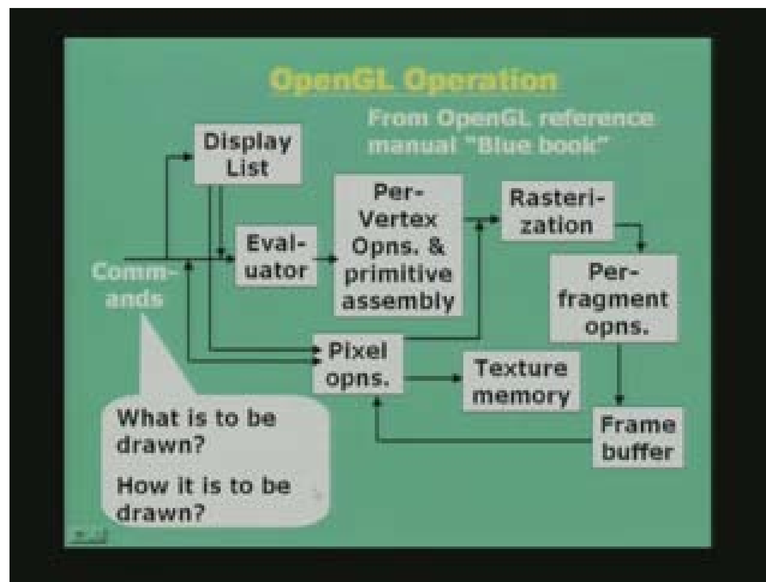
We talked of this alpha as a color parameter. If you look back into the slide in terms of this color parameter RGB which is addition and known to you as the three components of a color in a cube you have this parameter called alpha which is used for blending which tells you how much transparent an object is, is it opaque or is it transparent, is it transluscent or semi transparent or not. So OpenGL will tell you or gives you the option of providing whether the surface which you are drawing is completely opaque are not, completely transluscent or it is somewhere in between. So that is the alpha factor which you provide in a scale from 0 to 1 normalized. Of course you might have to give a value from 0 to 55 and tell you what is the blending alpha factor for a particular object surface or even at a high level however called any primitive may have its corresponding color not only in terms of RGB but also alpha.

The color of the light may not have an alpha factor, it will only have an RGB but the material may have or primitives will have an alpha factor. So coming to the last part of the features in OpenGL we talked of this atmospheric fog alpha blending corresponding to this alpha and you can have effects such as even motion blur. That will be very interesting, it will have concepts such as even motion blur. So these are the certain features which we had seen in OpenGL and based on these features we can start to discuss about the OpenGL graphics pipeline.

Finally let us read through these features again, 3D transformations, color models, lighting, rendering, modeling and other miscellaneous effects such as atmospheric fog, alpha blending and motion blur.

OpenGL commands are always processed in the order in which they are received. Although there may be an indeterminant delay before a command takes effect this means that each primitive is drawn completely before any subsequent command takes effect. It also means that state querying commends returns data that is consistent with the complete execution of all the previously issued OpenGL commands. And the effects of the OpenGL commands will be there on the frame buffer and it is ultimately controlled by the window system that locates the frame buffer resources.

(Refer Slide Time: 25:03)



The basic OpenGL operations will involve a few stages which we will see later on as we go ahead. This is a flow chart which will talk of an OpenGL operation. And this is picked up from the reference called OpenGL reference manual called the blue book. Of course there is a red book stated also but you can follow one of these for the OpenGL reference. So as you can see here there are lots of blocks through which the commands will go through and finally it will reach the frame buffer or texture memory. That is the final output which takes place. So when you writing an OpenGL program you are basically writing a set of associated commands. And these commands have to be interpreted. These commands first will be put into a display list.

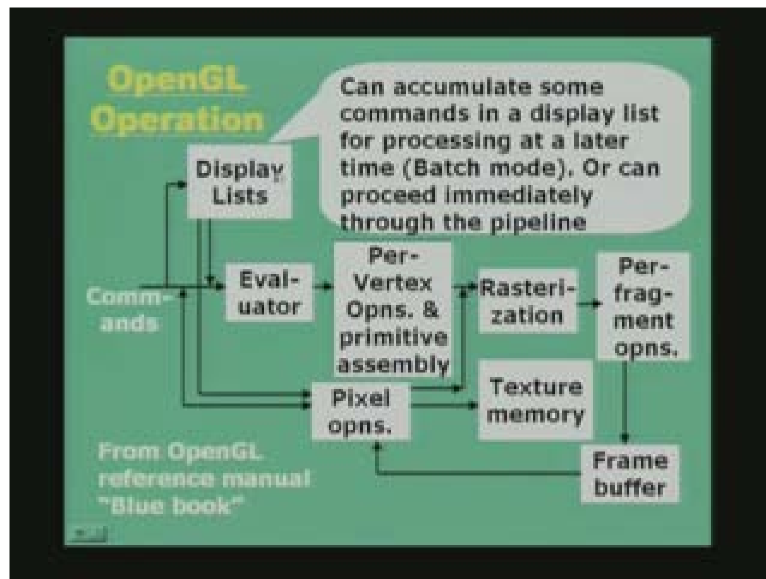
So what is this command? If you look back the commands will tell you what is to be drawn and how it is to be drawn.

So commands are, you would say I like to draw a set of polygons, those polygons could be numbers up to millions, a few millions, a few hundred millions for a more and very highly complex picture and where are those polygons? What are their attributes? How should I lead them up using a certain light sources? What should be the atmospheric fog and what is my view angle?

You specify all those in your commands and the OpenGL will take those commands and run in a particular sequence. Of course you can have a loop at a certain stage but still it is more or less sequential. Of course program in execution works only in a sequential mode we know that except when we talk about concepts such as parallel processing and synchronization. We talk of olden days program from Basic, FORTRAN, Pascal C, let us not bring in the concepts of threads and another paralleling concepts, it is a sequential program.

Sequential set of commands is what the OpenGL will look at and it will generate a display list. So let us look back, it will generate a display list and what is that display list. That display list can accumulate some commands for processing at a later stage, at a later timing batch mode or it can proceed immediately through the pipeline.

(Refer Slide Time: 00:29:55)



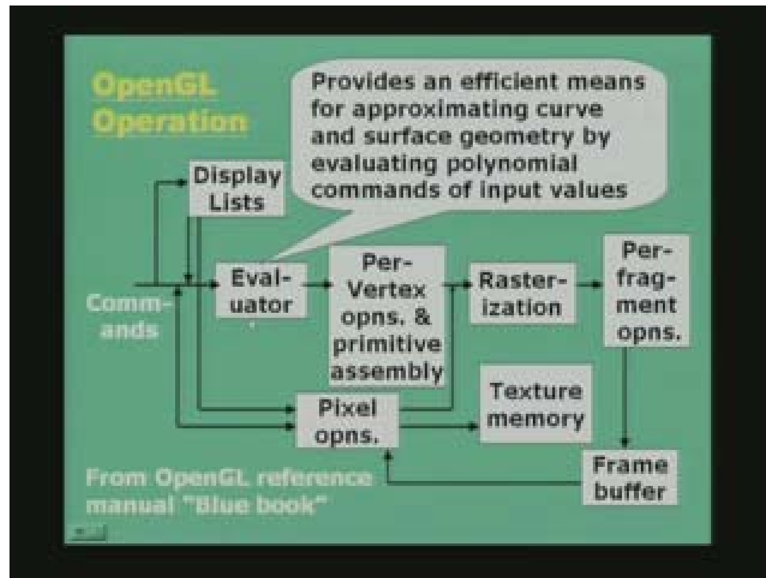
So you have these commands ready and OpenGL will generate the display list.

I hope you have not forgotten the concepts of display list which we discussed with display devices. The display list is the sequence of commands which are to be executed, it could be just draw line command, draw point command, draw primitives at some higher levels. But at the lower level it just becomes draw lower level primitives of OpenGL. And that display list is generated by OpenGL from your commands.

As you look back it can accumulate some commands in a display list for processing at a later time in batch mode or it can proceed immediately through the pipeline. And this is a pipeline, we will see a set of stages of the pipeline. You can see the different stages of the pipeline starting from evaluator, and a per-vertex operations and primitive assembly, rasterization, per-fragment operations.

So these are the four different stages of the pipeline which will generate operations on the frame buffer or generate a picture on the frame buffer or generate pixel operations for the texture memory. So the first stage in the pipeline which comes is the evaluator. What does the evaluator do? It provides an efficient means of approximating curve and surface geometry by evaluating polynomial commands of input values.

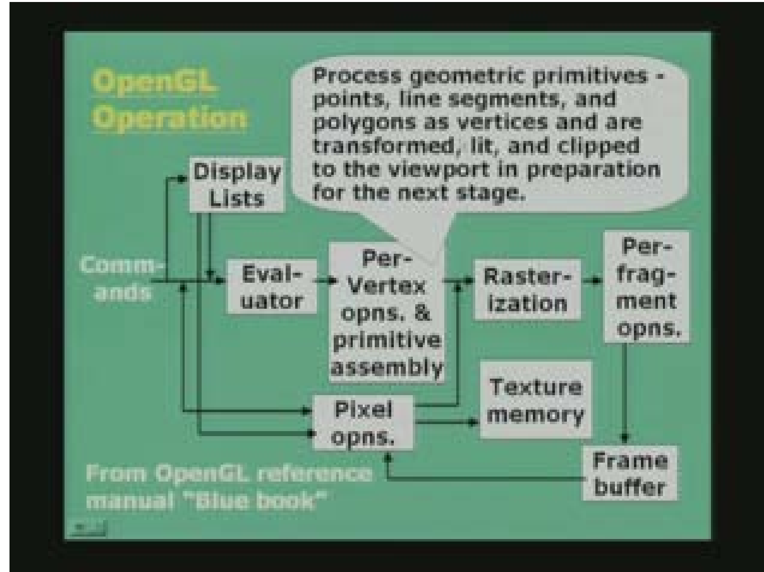
(Refer Slide Time: 00:31:21)



I will repeat; the evaluator provides an efficient means for approximating curve and surface geometry by evaluating polynomial commands of input values. The evaluator stage of processing provides an efficient means for approximating curve and surface geometry by evaluating polynomial commands as given here.

During the next stage of per-vertex operations we will see how the evaluator outputs are analyzed. The per-vertex operations and primitive assembly processes geometrical primitives. It processes points, line segments and polygons as vertices and are transformed lit and clipped to the viewport in the preparation for the next stage.

(Refer Slide Time: 00:32:04)



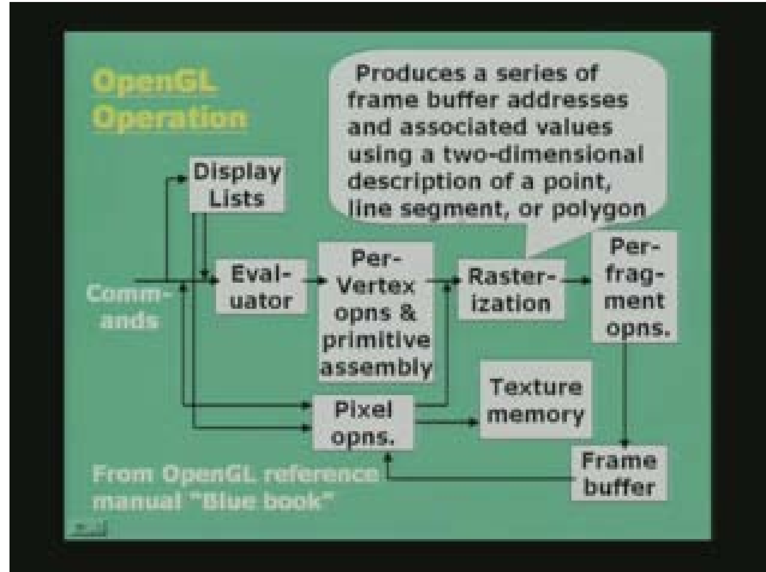
So, as you see the evaluator provides an efficient means, it takes the input from the command directly or from the display list and provides an efficient means for approximating curved segments and surface geometry by evaluating polynomial commands of input values. So you will have a set of polynomials which are evaluated here and the per-vertex will process geometric primitives.

So at this point we are only processing points, line segments and polygons as vertices and then they are transformed here if necessary based on the transformation matrix which we will talk about, we will open up this block later on and then they are lit depending upon the light sources and they are clipped if necessary to the viewport in preparation for the next stage. So as you can see here these first two boxes of the OpenGL pipeline, the evaluated and the per-vertex operations and primitive assembly, these two together form the different stages of the graphics pipeline which we have studied earlier in terms of the 3D viewing pipeline.

If you remember we had normalizing transformations, clipping to viewport and of course they generate a rendering of course it comes later on but when we are talking of those two stages mostly normalizing transformations and then clipping so OpenGL has similar stages in its own pipeline in terms of what you call as evaluators and per-vertex operations and primitive assembly. So if you look back we are we are talking about references from the OpenGL blue book where it talks of per-vertex operations and primitive assembly.

This block processes geometric primitives, points line segments and polygons as vertices and are transformed lit and clipped to the viewport in preparation for the next stage.

(Refer Slide Time: 00:34:13)

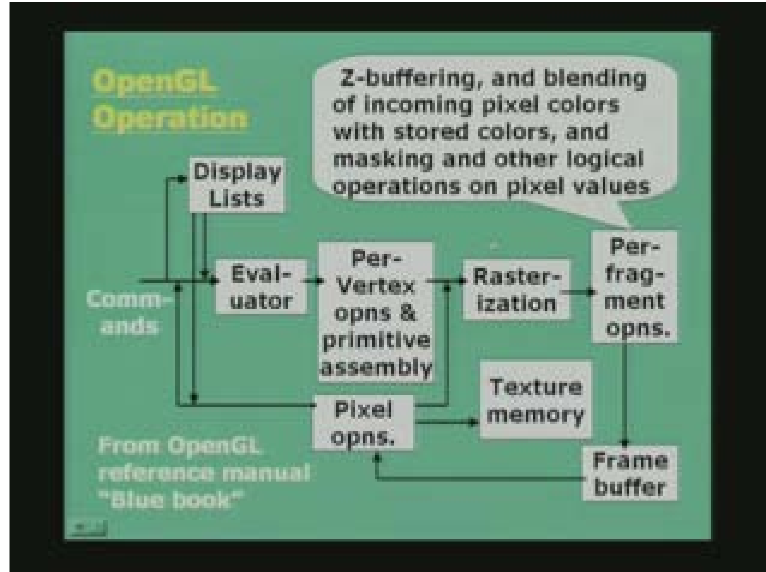


What does rasterization do? It produces a series of frame buffer addresses and associated values using a two dimensional description of a point line segment or polygon. So, this a process of visible surface, detection and shading which comes here where this rasterization produces a series of frame buffer addresses and associated values using a two dimensional description of a point line segment or a polygon. It is important to introduce the concepts of a fragment here which is very very important.

Now sometimes although we have seen so far that the OpenGL pipeline is processing the attributes in terms of points, lines, polygons and so on and it will do all the operations based on those primitives but there is a concept of fragment here. And with the output of this rasterization process goes to the fragment we will see what is this per-fragment operations first and then I will discuss what this fragment concept in OpenGL is.

Let us go back, here we are discussing about rasterization and the per-fragment operation does the concept of z buffering and blending of incoming pixel colors with stored colors and masking and other logical operations on pixel values.

(Refer Slide Time: 00:35:34)



Well, we had seen that rasterization produces a series of frame buffer addresses and its associated values are using a two dimensional description of a point line segment and polygon.

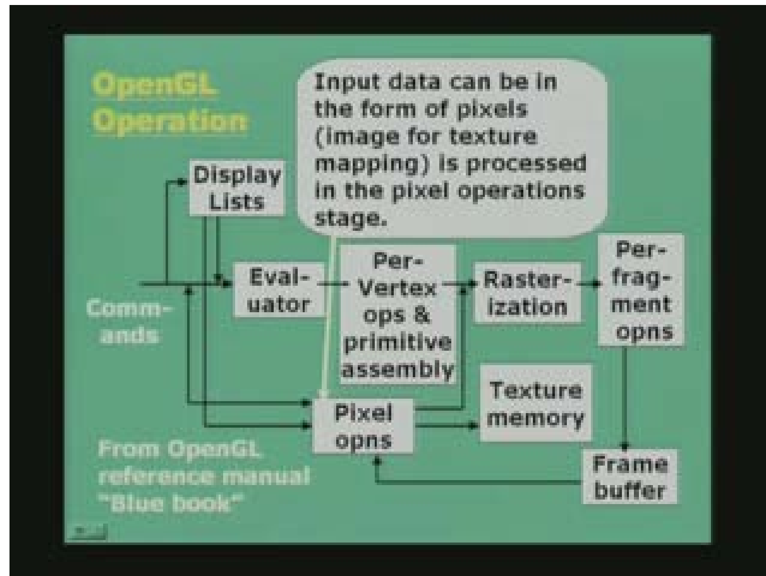
Each fragment which is produced now is fed into the last stage which is called the per-fragment operations which performs the final operations on the data before it is stored as pixels in the frame buffer here. So these operations include conditional updates to the frame buffer based on incoming and previously stored z buffer values for z buffering, blending of incoming pixel colors with stored colors as well as masking and other logical operations on pixel values. So the per-fragment operations will operate on the frame buffer with the concepts of z buffering, blending with pixel colors or stored colors a prairie, it uses the concept of masking and other logical operations on pixel values.

When you talk of logical operations on pixel values it could be a possibility that you can have a different material color and different light color and you have to superimpose one another. And then it is possible that you could like to change either the material color on line or position at the light color itself of the light source. And if you do that you do not want to redraw the entire picture because in that case the not only the redrawing process will be slow but you will also be cutting of the scope of animation.

So when you talk of these per-fragment operations at the end which is responsible form taking this rasterization outputs from the primitives being processed in the previous three stages of pipeline it will load on to the frame buffer that is the per-fragment operations load on some information to the frame buffer about each pixel related with this head buffering the color, the masking operations and logical operations with stored colors and also incoming pixel colors.

So it is a combination of these effects at the end which is latched on to the frame buffer and is handled by this per-fragment operation. This is what you see as the per-fragment operation.

(Refer Slide Time: 00:37:52)



What do you mean by these pixel operations?
We had seen this per-fragment operation, now the pixel operations mean the input data can be in the form of pixels that is the image of texture mapping and is processed for the pixel operation stage. So this pixel operation also can take input from the display list or it can set input or an output back on to the commands and it writes onto the texture memory.

So, input data can be in the form of pixels rather than vertices because it is possible that the commands which generate vertices in the display list can generate outputs to pixels so you may have an image data to process. Such data which might describe an image for use in the texture mapping skips the first stage of processing described data. That means it bypasses all these evaluator per-vertex operation and primitive assembly and rasterization and directly it goes to pixel operations. So it keeps all the first stage of processing and instead is processed as pixels in the pixel operations stage. So directly the image is brought here.

The result of this stage is either stored as texture memory for use in rasterization process or even you see a channel link to the process of rasterization itself or it could be rasterized and then resulting per-fragments or resulting fragments are merged into the frame buffer just as if they where generated from the geometric data.

Sometime back I said that OpenGL provides you to draw, simulate basically draw, you can simulate for engineering applications but it draws 2D and 3D pictures and digital images that is pixels. These are the three primitives.

When we talk of primitives of 2D and 3D objects I will say pixels in the form of digital pixels of a digital image also are a primitive. You can say; I would like to take this input image and do certain manipulations. And it is possible that you may not have any 2D or 3D objects in our picture.

You would like to have a background with the static two dimensional frames, a static two dimensional background with a digital image of a picture which you like that could be possible. Or you could have a surface on which you would like to map a particular digital image that is called as a texture mapping. You could have a texture itself or a simple in a digital image which you could map. So, depending upon that primitive request which is not a 2D or a 3D vertex which is a set of pixels, array of pixel values forming a digital image you would like to process only that, display only that, or only manipulate that or map it on to a surface.

So there are two such requests and based on these the processing stage in this case will bypass all the evaluator, per-vertex operations and primitive assembly, it will also bypass at least these two stages. It will bypass the evaluator it does not need the per-vertex and primitive assembly operations. You could pass this on to a texture memory directly or pass it on to a rasterized operation depending upon what you want.

If you want just the picture to be displayed with a little bit of manipulation you directly pump the pixel operations on to the texture memory. However, if you want this digital image to be mapped on to a surface then you need to pass it through the rasterization operation and through the per-fragment operation.

So as you can see in this picture from the commands and display list you have two separate channels coming into the pixel operation stage and the output could go to rasterization operation or texture memory depending upon the demand. Depending upon the task at hand which you want to solve the output or pixel operations might go to rasterization or texture memory. If it goes to rasterization it has to pass through per-fragment operations here and then drive it into the frame buffer because frame buffer typically will have all those vertices based on which you want to map this digital image.

You want to map this or lay over like you can visualize as if I am laying a cloth on hilly surface on a hilly model on an arbitrary corrugated surface as best as a sheet you can take or even a corrugated surface you want to map a particular texture cloth. And you want to do that even that operation can be done with the help of OpenGL primitive through the rasterization process and per-fragment operations in it because already in the frame buffer you have the per-fragment operations which is working with those primitives and you want to put this map instead of putting the color.

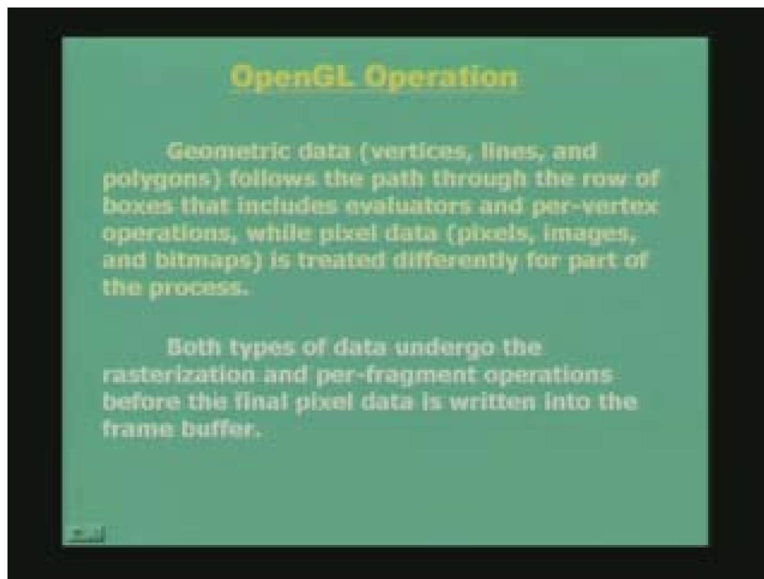
You want to put this image as a map laid on the map it is something like you take the map and put it on a sphere that becomes your globe. If you take a two dimensional image of the world map and try to wrap it around a sphere it becomes a globe. So that is the

simple example of an operation of what I mean by taking a digital image and processing it on a certain primitive.

Your primitive is a sphere and a two dimensional digital image of the world map could be your digital image or the pixel which you are operating on. So the operation requires that you take this map and encapsulate or wrap around this sphere to generate your globe. So that is distinct, it is very much possible with OpenGL where you use the link to the rasterization operation, per-fragment operations and frame buffer and the input data in the pixel operation.

Again I repeat; it could be in the form of pixels which is a digital image for texture mapping and is processed in the pixel operation stage.

(Refer Slide Time: 00:44:06)

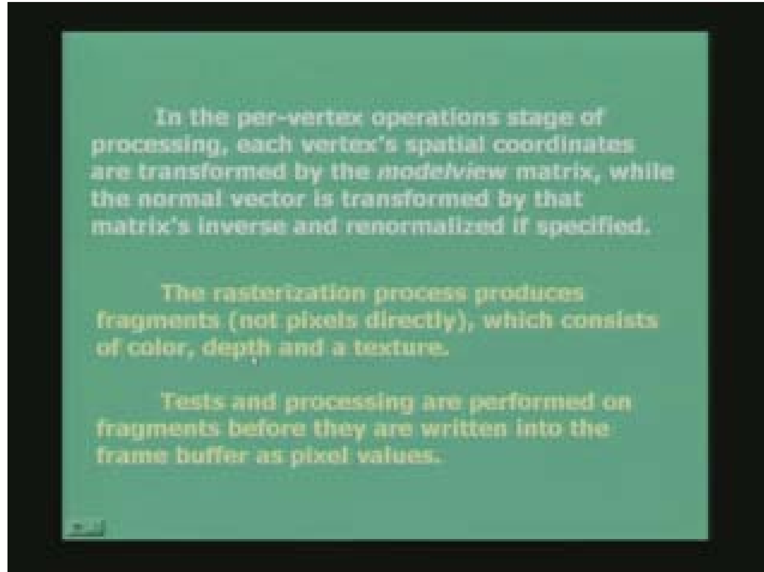


So when we discussed about OpenGL operation we are talking of geometrical data which follows in the form of vertices, lines and polygons which follows the path through the row of boxes which we have shown in the previous slide that includes evaluators and per-vertex operations while the pixel data it could be in form of pixels, images and bitmaps is treated differently for a certain part of the process that is treated differently.

For both type types of data we undergo the rasterization and per-fragment operations before the final pixel data is written onto the frame buffer. In the per-vertex operations stage of processing each vertex, vertices, special coordinates are transformed by the modelview matrix.

We will come to this modelview matrix later on which is a part of the evaluator and the per-vertex operations while the normal vector is transformed by that matrices inverse and renormalized if specified.

(Refer Slide Time: 00:45:01)



We will talk about this in detail later on, we will skip this part.

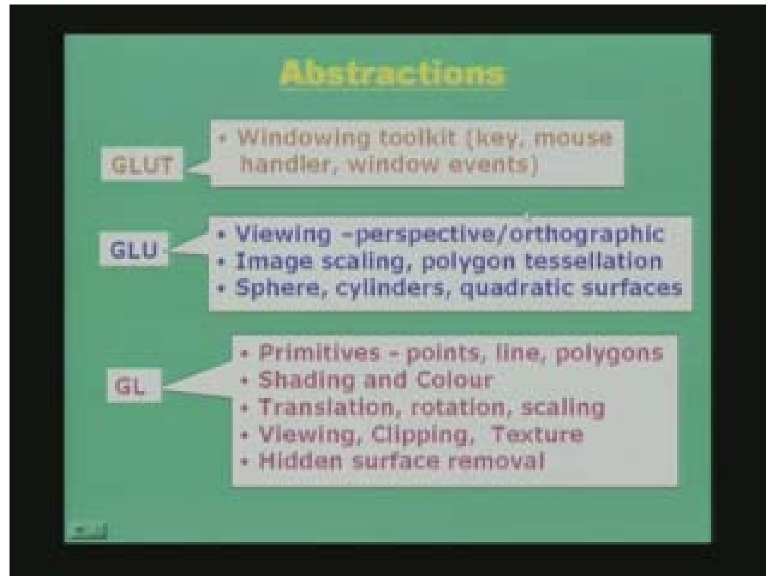
The rasterization process produces fragments not pixels directly which consists of color depth and a texture so that is a definition of the fragment which will say that fragments is not pixels but it consists of a set of pixels with color, depth and a particular texture. And tests and processing, various tests and processing are performed on these fragments before they are written onto the frame buffer as pixel values. So the per-fragment operation will do a lot of processing and tests on these fragments before they are written onto the frame buffer as pixel values. This is the overall description of the various stages of the OpenGL pipeline.

You start with primitives again to give a brief summary and these primitives could be in the form of 2D or 3D vertices or simple digital pixels. And depending upon the requirement the commands will generate a display list or pixels for the pixel operations and there could be two different channels for processing. One through the evaluator, per-vertex operation, primitive assembly, length to the rasterization and per-fragment operation or the pixel operations could go directly on to texture memory for simple pixel operations, image based manipulation which could be set of image processing tasks, which could be viewing this image with a certain contrast enhancement or remove noise, remove blurring and those effects where image processing operations could be done on.

And the other channel could be that these pixel operations are passed through the rasterization process and the per-fragment operations to work on the frame buffer directly along with this fragments which consists of information about color, depth and texture information coming out of this pixel operation so that is the brief description about OpenGL. I would love to spend more time on this OpenGL pipeline but we have to discuss few more aspects at least today before we take up OpenGL programming

examples in the next class. We talked about abstractions; let us look at abstractions of OpenGL.

(Refer Slide Time: 00:49:26)



If you look into the slide the graphical library for OpenGL will consist of primitives, points, lines and polygons. It can provide shading and color, it can provide translation, rotation and scaling.

It can provide viewing, clipping and texture, it can also provide a hidden surface removal. These are some of the feature in a graphics library. It provides you primitives to draw points, lines and polygons, it can provide concepts of shading and color. It can provide you options of matrix based transformations 2D and 3D which we have seen earlier. It can provide view clipping and texture, it can provide hidden surface removal. These are the low level facilities available in OpenGL, minimum for any OpenGL libraries.

Let us look into the GLU Graphics Library Utilities. It provides you high level viewing such as perspective and orthographic projection. It provides you image scaling, polygon tessellation, it provides higher level structures such as spheres, cylinders and quadratic surfaces. This will be very handy.

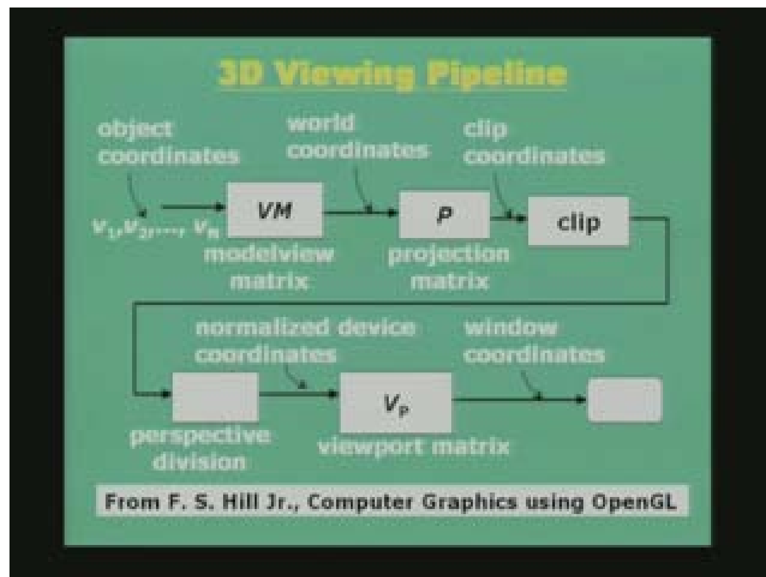
We have discussed about wireframe diagrams, sweep representations to generate certain primitive structures such as spheres, cones, cylinders or even surfaces with the help of curves. We discussed about parametric surface and all that. Now GLU provides you these as direct primitives. It is a big advantage that you do not need to spend time and write your own program and the program may not need time to generate these. So now you can treat these spheres, cylinders as an example and quadratic surfaces as primitives. And you do not need to worry about where those x y z are and how many precision you need to give, but that of course you have to decide.

But you did not to know to come up with the sweep representation to generate your x y z vertices for your sphere. So that is what GLU provides. GLUT is basically a windowing toolkit for interfaces for keyboard, mouse handler and other windows events. So GLUT is basically an interface to windows. And of course there is a whole lot of other operations with the GLU and GL did not provide, it is not in the list here. You must go to the OpenGL reference manual to come up with a list of such flexibilities which are provided.

If you look back I was just talking about translation, shading, color, primitives and all that and along with the operations to create vertices which may have certain flags, colors, matrix transformations, lighting and color. You can generate textures, you can generate various primitives, various types of clippings are possible, provisions are provided, transform to window coordinates, rasterizations, you can generate pixels, generate bit maps, texture memory and then of course you have the concepts of fragments.

We will discuss about the concept of Phong, about atmospheric attenuation, about the concept of test which are done on fragments such as seizer test, alpha test, stencil test, death buffer test, blending, dithering, logical operations on pixels, copying pixels operations, frame buffer operations on pixels and so on and so forth. These are various types of operations which are possible in the OpenGL and GLU. Now it is not possible in a short frame of time to discuss each of these commands because this is not a lecture series only on OpenGL. This is just to kick start you on OpenGL so that you get excited and will start to use it. Of course you see some examples about this programs really working on my system you will really get excited and will be able to see how it works. So coming back to abstractions we have this windowish toolkit of GL, GLU and GLUT.

(Refer Slide Time: 00:51:18)



Well, we talked of a 3D viewing pipeline which is also present in your OpenGL environment where we have a 3D viewing pipeline which starts with object coordinates transformed by the modelview matrix to the world coordinates.

And the modelview matrix V into M which you can set will transform this object coordinates to world coordinates then the projection matrix will transform the world coordinates on to clip coordinates.

Then you can have a clipping stage which will take the input of this clip coordinates and generate and pass it on to the perspective division matrix and that will generate the normalized device coordinates.

We discussed about this canonical view normalized device coordinates earlier and then of course we have the viewport matrix which will generate the window coordinates in output or written on to the device port.

I must inform you that this flow chart is taken from the book by F. S. Hill Junior which is computer graphics using OpenGL, computer graphics using OpenGL by F. S. Hill. I will give you a detailed list of references later on in the next class after I finish the coverage of OpenGL. But this 3D viewing pipeline is different from the OpenGL pipeline which we discussed in terms of the architecture of OpenGL that was different. This is a 3D viewing pipeline which is similar to the concept of a viewing pipeline which we discussed earlier.

We discussed this viewing pipeline in the earlier class when we discussed 3D transformations and viewing. So you can see that 3D viewing pipeline of OpenGL is exactly the same or almost similar to what we discussed earlier. So these coordinates which are passed on, if you see here the objects coordinates passed on by the modelview matrix generate world coordinates, projection matrix, generate clip coordinates after clipping and perspective division generate the normalized device coordinates then a viewport matrix will generate the window coordinates which you put on to the viewport. This is the overall viewing pipeline and just to windup what we discussed today we started with OpenGL then we discussed the features of OpenGL, we discussed about the various stages of the OpenGL architecture as such and its features and finally the 3D viewing pipeline.

Not all the features which we discussed today here but I just try to give you as much of the list of features at least orally if not by the full list. We will discuss a few of this when we get into the code of the OpenGL later on the next class. But remember, OpenGL is a state machine that is what you must also remember and you can put it into various states, the states may not intermix with each other but sometimes you may be careful when they are interacting. The color of one of the light might interact with the color of the material. It might interact with certain animation stages which you are putting through these vertices. And you can put it to a static state into a motion state both in terms of the drawing pictures and then manipulating these pictures.

With these words we would like to wind up the discussion on the first lecture on OpenGL. In the next lecture we will move on to further few more features about OpenGL and basically we will open up with the structure about the GLUT program.

Time is not permitting me to discuss about the structure of OpenGL program right now but I will discuss that. We will follow the GLUT structure program based on c plus plus syntax. So please go back and look into c plus plus features, may not be object orientation to start with but we will follow the syntax of c plus plus programs and then move on to certain examples and functions. And I hope you will be delighted to see some OpenGL programs really working on my machine, thank you very much.